|  |
| --- |
| Budapesti Műszaki és Gazdaságtudományi Egyetem |
| Diplomamunka |
| [Document subtitle] |

|  |
| --- |
| Bujtor Bálint  [Date] |

Table of Contents

[1. Abstract (3-4 paragraphs): 1](#_Toc56077952)

[1.1. Subject placement, importance of topic 1](#_Toc56077953)

[1.2. Quick summary of what was done and why 1](#_Toc56077954)

[1.3. How it turned out 1](#_Toc56077955)

[2. Introduction (2-3 pages): 2](#_Toc56077956)

[2.1. More detailed introduction to the broad topic 2](#_Toc56077957)

[2.2. Intro into the narrow topic 2](#_Toc56077958)

[2.3. Problem statement (what is important/unsolved that this thesis addresses) 3](#_Toc56077959)

[2.4. Short summary of proposed solution(s) 3](#_Toc56077960)

[2.5. Short summary of results 4](#_Toc56077961)

[2.6. Quick overview of the thesis structure 4](#_Toc56077962)

[3. Related Work (15-20 pages): 5](#_Toc56077963)

[3.1. Give introduction into broad field (e.g: computer vision, deep learning) 5 pages 5](#_Toc56077964)

[3.2. Give introduction into narrow field (e.g: object detection, reinforcement learning) 5 pages 6](#_Toc56077965)

[3.3. Describe chosen algorithm and its alternatives 5 pages 6](#_Toc56077966)

[3.3.1. Make sure you do some pros/cons 6](#_Toc56077967)

[4. Specification (4-5 pages): 6](#_Toc56077968)

[4.1. Describe the actual task in detail 6](#_Toc56077969)

[4.2. Make sure to mention constraints, limitations, requirements 6](#_Toc56077970)

[5. Design (chapters 5-6-7 should be 20-30 pages): 6](#_Toc56077971)

[5.1. Which of the algorithms described in the Related Work chapter we chose and why 6](#_Toc56077972)

[5.2. What frameworks are available and how are they different and why we chose the one we chose 6](#_Toc56077973)

[5.3. Software design (if there is one) 6](#_Toc56077974)

[6. Development: 6](#_Toc56077975)

[6.1. What Programming language/IDE/Environment we used 6](#_Toc56077976)

[6.2. Dataset creation (if there is one) 6](#_Toc56077977)

[6.3. Describe implementation of modules 6](#_Toc56077978)

[6.4. Describe algorithms/neural net structure 6](#_Toc56077979)

[6.5. Describe different attempts 6](#_Toc56077980)

[7. Experimental Results: 6](#_Toc56077981)

[7.1. Describe experiment methodology 6](#_Toc56077982)

[7.2. Describe test cases 6](#_Toc56077983)

[7.3. Describe metrics used for evaluation 6](#_Toc56077984)

[7.4. Describe dataset (if there is one and it wasn't created) 6](#_Toc56077985)

[7.5. Describe test cases 6](#_Toc56077986)

[7.6. Evaluate results 6](#_Toc56077987)

[8. Conclusion (1-2 pages): 6](#_Toc56077988)

[8.1. Give general conclusion (what we did and why is it good) 6](#_Toc56077989)

[8.2. Evaluate results quickly 6](#_Toc56077990)

[8.3. Describe opportunities for further research/improvement 6](#_Toc56077991)

# Abstract (3-4 paragraphs):

Szakdolgozatom témáját a mesterséges intelligencia területéről, azon belül a deep learning témájából írtam. Feladatnak azt választottam, hogy lehetséges-e egy valós környezetben is futtatható objektum detektáló algoritmust mesterségesen szintetizált képeken tanítani, majd pedig a betanított algoritmust valós környezetben sikeresen tesztelni, alkalmazni.

## Subject placement, importance of topic

A mesterséges intelligencia napjaink egyik igen felkapott témája, rengeteg kutatás történik ezen a tématerületen, számos és egyre több szakterületen alkalmaznak sikeresen mesterséges intelligencia alapú algoritmusokat, szolgáltatásokat. Az feladatom is ehhez a népszerű témakörhöz tartozik. A témával foglalkozó kutatások az utóbbi években egyre inkább kezdenek megjelenni a mindennapi életben, gondoljunk az egyre nagyobb autonómiával bíró vagy akár teljesen autonóm járművekre. Ezek mindennapi jelentősége hatalmas. Egy biztonságosan működő autonóm jármű esetén a közúti balesetek száma jelentős mértékben csökkenthető, hiszen a számítógép vezérlése által elimináljuk az emberi mulasztásból, hibából fakadó baleseteket. (ESETLEG VALAMILYEN GRAFIKON, KÖZÚTI BALESETEK SZÁMA, EMBERI MULASZTÁS MÉRTÉKE, MINT OKOZÓ)

## Quick summary of what was done and why

Az én munkám egy mesterséges intelligencia alapú algoritmus megalkotása volt, amely képes valós időben objektumokat detektálni. A mesterséges intelligencián alapuló algoritmusok sikeres betanításának alapja a megfelelő minőségű és pontosságú adatbázis megalkotása. Ebből kifolyólag egy új feladat során ennek az adatbázisnak az előállításával telik el a legtöbb idő. Az én feladatom az adatbázis előállításával eltelő időt eliminálja azáltal, hogy olyan szintetikusan generált képeken tanítja be a mélytanuláson alapuló algoritmust, aminek segítségével az adatbázis generálás automatikusan, emberi felügyelet nélkül történik meg. Ezáltal nagy mértékben rövidíteni lehet egy új mesterséges intelligencia alapú algoritmus kifejlesztését. A feladatom során valós időben kell detektálnom embereket, járműveket, közúti táblákat, illetve jelzőlámpákat.

## How it turned out

# Introduction (2-3 pages):

A következő néhány oldalon egy rövid bevezetőt fogok adni a tématerületről, részletesebben bemutatom a feladatot. Ki fogok térni arra is, hogy pontosan milyen problémára kellett megoldást találnom, valamint arra is, hogy ezt milyen eredménnyel sikerült elvégeznem.

## More detailed introduction to the broad topic

A szakdolgozatom témájának területét a számítógépes látás és a mesterséges intelligencia alapú rendszerek alkotják. A számítógépes látórendszerek olyan algoritmusok segítségével oldanak meg problémákat, amelyeknek bemenete egy adott kép vagy képsorozat, kimenete pedig valamilyen magasabb absztrakciós szintű információ vagy egy módosított kép, amely a felhasználó számára valamilyen szempontból előnyösebb. Utóbbi esetben képfeldolgozásról beszélünk (image processing), előbbi esetben pedig számítógépes látásról (computer vision). A számítógépes látást gyakorta két csoportra szokták osztani, a hagyományos látásra és a gépi tanuláson alapuló algoritmusokra (machine learning). Ezeket a módszereket szokás mesterséges intelligencia alapú (artificial intelligence – AI) algoritmusoknak nevezni.

## Intro into the narrow topic

Az utóbbi években jelentős mértékben nőtt a népszerűsége az úgynevezett mélytanuláson (deep learning) alapuló számítógépes rendszereknek. A mélytanulás a gépi tanulás egyik módszere. A technika lényege, hogy azt tanítjuk meg a számítógépnek, amit mi emberek természetesen csinálunk. Példákon keresztül tanulunk. A mélytanulás olyan technológiák alapja, mint az önvezető autók, hangfelismerő rendszerek (Shazam) vagy akár tőzsdei elemzőprogramok.

A deep learning segítségével olyan komplex problémák megoldása vált lehetségessé, amelyeket a hagyományos számítógépes látáson alapuló programokkal egyáltalán nem, vagy csak óriási erőfeszítések árán sikerült volna megoldani. Az ilyen algoritmusok előnye, hogy segítségükkel olyan problémákat is meg lehet oldani, amelyeknek megoldását magunk sem ismerjük. Ezek a problémák általában magas absztrakciós szintűek. A deep learning alapú algoritmusok hátránya, hogy maga az algoritmus egyfajta feketedobozként funkcionál, nem teljesen tudjuk, hogy pontosan milyen folyamatok játszódnak le benne, hogyan jutott el az algoritmus oda, hogy megfelelő konfidenciával megoldjon egy adott feladatot. A mélytanuló algoritmusok rendszeresen alkalmaznak numerikuson optimalizálást vagy statisztikai módszereket, emiatt nem lehetnek 100%-an pontosak.

A mélytanuló algoritmusokat többféleképpen lehet csoportosítani. Az egyik legelterjedtebb felosztás szerint megkülönböztethetünk felügyelt tanítást, felügyelet nélküli tanulást, valamint megerősítéses tanulást. Mindegyik csoportnak megvannak az előnyei és hátrányai is, éppen ezért más-más területeken terjedtek el sikeresen.

A mélytanulás legegyszerűbb fajtája a felügyelt tanulás, amelynek során minden egyes bemenet mellé tartozik egy elvárt kimenet is. Ilyenkor azt várjuk el, hogy az algoritmus minél nagyobb arányban, minél pontosabban tanulja meg, hogy egy adott bemenethez mi a megfelelő, elvárt kimenet. Ennek a tanulási módszernek azonban számos korlátja, gátja van. A felcímkézett tanító adatbázis előállítása rendkívül költséges, időigényes feladat, minősége pedig nagymértékben meghatározza a tanítás sikerességét is. Másik hátránya pedig az, hogy csak olyan problémákat tudunk ezekkel az algoritmusokkal megoldani, amelyeknek a megoldását mi magunk is ismerjük.

Létezik felügyelet nélküli tanulás is, amelynek során a tanuló algoritmus számára csak bemeneti adatokat szolgáltatunk, elvárt kimeneteket nem adunk meg. Ilyenkor a cél az, hogy az algoritmus valamilyen séma alapján maga alkossa meg a kimenetet, valamilyen kompakt módon. Előnye ennek a módszernek, hogy az adatbázist rendkívül olcsón, akár automatikusan is elő lehet állítani, valamint olyan problémák is megoldhatók vele, amelyeknek megoldását mi sem ismerjük. HÁTRÁNY??

A gépi tanulás harmadik fajtája a megerősítéses tanulás (reinforcement learning). Az előbbi két módszertől ez több dologban is különbözik. A megerősítéses tanítás során az esetek döntő többségében összefüggő döntéseket, döntéssorozatot kell meghozni és a döntések helyességéről nem minden esetben kap visszajelzést. A másik különbség a visszajelzés mibenlétében rejlik. Ez a visszajelzés csak azt mondja meg, hogy mennyire volt jó az adott döntés, azt nem mondja meg, hogy mi lett volna a helyes döntés. A megerősítéses tanulás alkalmazási területére egy remek példák a különféle játékok (sakk, számítógépes játékok) vagy a járműirányítási feladatok.

## Problem statement (what is important/unsolved that this thesis addresses)

A szakdolgozatom témája a felügyelt tanulás köre a mélytanuláson belül. Mint azt az előző hasábokon említettem, a felügyelt tanítás egyik nagy hátránya, hogy a tanító adatbázis előállítása és felcímkézése egy rendkívül költséges, időigényes folyamat. MÉG MIT LEHET??

## Short summary of proposed solution(s)

Az én munkám erre a problémára kínál egy megoldást. A munkám során ezt a tanító adatbázist szintetikus, számítógépesen generált képekből hozom létre, valamint az egyes képek felcímkézését is automatikusan végzem el. A cél egy olyan algoritmus létrehozása, amely képes egy járművön futó beágyazott környezetben, valós időben objektumokat detektálni. A közúti környezetben használt mélytanuló algoritmusokat leginkább olyan problémák megoldására szokták alkalmazni, amelyek valamilyen módon javítják a vezető komfortérzetét, növelik a biztonságot vagy terhet vesznek le a vezetőről. Ilyen probléma lehet az automatikus vészfékezés, ha például egy gyalogos vagy biciklis halad át a jármű előtt. További példa lehet az automatikus távolságkövetés, illetve a tábla- és jelzőlámpa felismerés és ezáltal a közúti szabályok betartása. Éppen ezért munkám során azokat az objektumokat igyekeztem betanítani az algoritmusomnak, amelyek a fent bemutatott problémák szempontjából elengedhetetlenek. A mélytanuló algoritmusomat a következő objektumok felismerésére tanítottam be: gyalogos, jármű közlekedési tábla és jelzőlámpa.

Fontos szempont volt a feladatot és az algoritmust tekintve, hogy valós időben fusson ezáltal a mindennapi környezetben, közúton is használni lehessen. Olyan algoritmust kellett választanom és használnom, ami valós időben dolgozza fel a beérkező adatokat. Enélkül nem lehetne garantálni éles környezetben, hogy megfelelő időn belül képes lefutni és döntést hozni az algoritmus.

Többször említettem azt is, hogy az algoritmusnak a valós életben is meg kell állnia a helyét. Ezért azt is meg fogom vizsgálni a munkám során, hogy a betanított algoritmus, hogyan szuperál valós környezetben készült képeken. Ezért egy olyan adatbázison fogom tesztelni az algoritmusomat, amely ilyen képeket tartalmaz.

## Short summary of results

## Quick overview of the thesis structure

A bevezetés lezárásaként be szeretném összefoglalni, hogy a következő fejezetekben miket fogok bemutatni. Elsőként részletesen be fogok számolni a munkám tématerületéről, ami a mesterséges intelligencia és a számítógépes látás. Be fogok számolni arról is, hogy a mesterséges intelligenciának milyen módszerét használom a munkámhoz. Kitérek ennek a területnek a bemutatására, a jelenlegi kutatásokra, a legújabban használta metódusokra és módszerekre.

Miután részletesen bemutattam a tématerületemet rátérek a saját munkám részletezésére. Elsőként specifikálom a pontos feladatot és bemutatom a tervezés egyes lépéseit. Ezt követően rátérek a fejlesztésre, részletesen bemutatom az egyes lépéseket, az esetlegesen felmerült problémákat és a megoldásukat. Végül pedig bemutatom az eredményeket.

# Related Work (15-20 pages):

Jelen fejezetben részeletesen be fogom mutatni azt a szakterületet és tématerületet, amit a szakdolgozatom és diplomamunkám végzése során megismertem. Elsőként a szélesebb, általánosabb bemutatót fogok adni a témáról, majd rátérek arra tématerületre, ami szorosabban kapcsolódik a szakdolgozatmunkámhoz.

## Give introduction into broad field (e.g: computer vision, deep learning) 5 pages

Mint ahogy azt már említettem az eddigiekben, a szakdolgozatom témájának gerincét a számítógépes látás és a mesterséges intelligencia alkotja. Az én feladatom ennek a két tématerületnek a mezsgyéjén helyezkedik el, ezért mindkét tudományterületet meg kellett ismernem és először ezt a két területet fogom ismertetni.

### Computer vision

Elsőként a számítógépes látás témáját szeretném röviden bemutatni. A számítógépes látás olyan feladatokkal, problémákkal foglalkozik, amelyek során valamilyen vizuális, képi adatból szeretnénk információt kinyerni és ezt felhasználni. Ez az információ sokféle lehet. Egy adott képből kinyerhetünk távolság adatokat, megszámolhatjuk, hogy egy objektumból mennyi van az adott képen, megszámolhatjuk, hogy hányféle objektum van. A lehetőségek gyakorlatilag korlátlanok. A számítógépes látás célja, hogy ezeket az adatokat képekből, képsorozatokból algoritmikus eszközök segítségével kinyerjük egy másik döntéshozó alkalmazás vagy személy számára. A számítógépes látás egyik legnagyobb kihívása az, hogy óriási (akár milliós) adathalmazzal dolgozik és ezeknek az adatoknak gyakorlatilag végtelen féle kombinációja állhat elő. Éppen ezért ezeknek a problémáknak a megoldásához hatékony algoritmusokra van szükség. Sokáig a hatékonyság, illetve a nem megfelelő számítási kapacitás szabott gátat a számítógépes látórendszerek széleskörű elterjedésének.

A számítógépes látás további nehézsége, hogy mi emberek egy látott kép alapján könnyen tudunk információt kinyerni, ugyanakkor az a folyamat, amelynek során ezt az információt kinyerjük a képből részben vagy akár teljesen tudat alatt történik. Emiatt ezeket a képességeket nehéz algoritmikus lépések sorozatává konvertálni. Ezen felül sokszor más érzékszerveinket is felhasználjuk a döntésünk meghozatalához, amelyet egy számítógépes látórendszer nem tud igénybe venni. Ezen problémák miatt a számítógépes látás területén sokszor használunk heurisztikus megoldásokat, gépi tanulásra vagy optimalizálásra épülő megoldásokat.

A számítógépes látó algoritmusokat sokféleképp szokás csoportosítani, mindazonáltal a legelterjedtebb megoldás az eljárások kimenete alapján történik. Eszerint megkülönböztethetünk képfeldolgozó (image processing) valamint a számítógépes látás (computer vision) algoritmusait. Előbbi esetében a kimenet egy olyan módon feldolgozott kép, ami a felhasználó szempontjából előnyösebb tulajdonságokkal rendelkezik. Ezen algoritmusok kimenete általában valamilyen további algoritmus bemenetét szolgáltatják, esetleg az emberi láthatóságot javítják. Utóbbi esetben olyan kimenetet kapunk, amely magasabb absztrakciós szinttel rendelkező információt szolgáltat. Ilyen információ lehet a képen lévő objektumok száma, színe mérete, de bármi egyéb is kinyerhető, amit szeretnénk.

KELL MÉG INFÓ? MIROL?

A számítógépes látáson belül legtöbbször két csoportot lehet megkülönböztetni, attól függően, hogy milyen módszereket használ fel az információ kinyerésére. Megkülönböztethetjük azokat a módszereket, amelyek a gépi tanuláson (machine learning) alapszanak, valamint azokat a módszereket, amelyek nem használnak gépi tanuló algoritmusokat. Ezeket hagyományos látó algoritmusoknak nevezünk. A következőkben a gépi tanuláson alapuló algoritmusokat fogom bemutatni, mert ilyen algoritmusokat alkalmaztam a munkám során én is.

HASZNÁLJAK ANGOL SZAVAKAT VAGY LECSERÉLJEK MINDENT MAGYARRA?

### Machine learning

A gépi tanulás a számítógépes algoritmusoknak olyan vizsgálata, kutatása, amelynek célja az, hogy olyan algoritmusok jöjjenek létre, amelyek képesek a tapasztalás segítségével fejlődni. [1] A gépi tanulást szokás a mesterséges intelligencia egyik részhalmazának is tekinteni. A gépi tanuló algoritmusok egy adott adathalmaz alapján készítik el az adott problémát megoldó modellt. Ezt az adathalmaz tanító adatbázisnak is szokás nevezni. Ezek a modellek olyan bemenetek alapján is tudnak döntést hozni, előrejelezni, amely bemenetre azelőtt nem voltak felkészítve. A gépi tanulást számos területen tudják használni, mint például az e-mailek szűrésénél vagy pedig a számítógépes látás területén is, mint ahogy azt később be is fogom mutatni.

A gépi tanulás során a cél az, hogy anélkül tanítsunk meg egy adott feladatra az algoritmust, hogy az adott feladat megoldását explicit módon megadtuk volna az algoritmusnak. [2] Egyszerűbb feladatok esetén ugyan könnyen meg lehet mondani az algoritmusnak, hogy mi az elvárt kimenet, hogy hogyan kell működnie, ugyanakkor összetettebb feladatok esetében ez nehézkes lehet. Gondoljunk csak arra, amikor egy videósorozatból szeretnénk információt kinyerni. Egy videó másodpercenként több mint húsz képkockát tartalmaz. Ha minden egyes képkockára meg szeretnénk mondani, hogy mi a helyes kimenet, akkor már egy néhány perces videó esetén is rengeteg időt pocsékolunk el. Ehelyett a gépet segítjük, hogy ő maga alkossa meg az algoritmusát az átadott bemenetek alapján. Azt, hogy a gép kialakítsa a saját algoritmusát sokféleképpen el lehet érni. Legtöbbször ezen megközelítések szerint szoktuk csoportosítani a gépi tanuló algoritmusokat.

Olyan esetekben például, amikor számos potenciális kimeneti lehetőség létezik, egy lehetséges megoldás az, hogy a bemenetek adott részét érvényesként, a többi részét pedig nem érvényéként címkézzük fel. Erre a módszerre a leggyakrabban hozott példa a számjegyfelismerés. Ebben az esetben a számítógép célja, hogy megtanuljon felismerni kézzel írott számjegyeket. Ilyen kézzel írott számjegyeket tartalmaz az MNIST adatbázis.

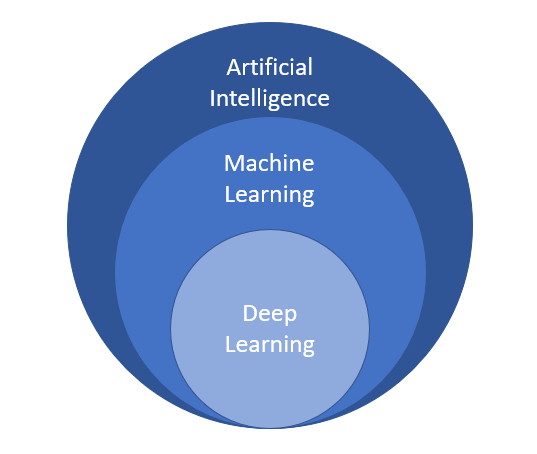
A gépi tanulást megközelítő módszereket leginkább három nagy kategóriába szokás csoportosítani, aszerint, hogy milyen természetben található meg jel vagy visszajelzés a tanuló algoritmus számára. Ezeket a kategóriákat már korábban bemutattam, így most csak ismétlés szintjén ejtek szót róluk. Beszélhetünk felügyelt tanulásról, amikor az algoritmus számára egy valaki más által előre felcímkézett adatbázist szolgáltatunk a helyes kimenetekkel. Az cél az, hogy ezen adatbázis segítségével az algoritmus kialakítson egy olyan általános szabályt, amelynek segítségével címkézetlen bemenetekre is nagy százalékban meg tudja adni a kimenetet. A nem felügyelt tanulás során csak egy adatbázist adunk meg az algoritmusnak, elvárt kimenetet nem. Ekkor a célunk az, hogy valamilyen mintát találjon az algoritmus az adatokban. A harmadik kategória a megerősítéses tanulás. Ennek során a program egy dinamikus környezettel érintkezik és a célja az, hogy elérjen valamilyen célt. Minden egyes döntés után kap egy visszajelzést arról, hogy milyen mértékben volt jó a döntése. Ez egyfajta díj, a program pedig ezt a díjat maximalizálja. [3] Természetesen vannak olyan megközelítések is, amelyek nem sorolhatók be a fenti három kategória egyikébe sem vagy egyszerre több kategóriát is alkalmaz a módszer. Ilyen be nem sorolható algoritmus például a dimenzió redukció.

A gépi tanulást, mint fogalmat először az IBM egyik munkatársa, Arthur Samuel használta, aki maga is egyik úttörője volt a témának. [4] Az érdeklődés a mintázatok felismerésének kutatásában az 1960-as években is folytatódott. Ekkor jelent meg Nilsson Learning Machines [5] című könyve, ami az időszak reprezentatív munkája volt és főleg a gépi tanulás mintázat felismerési lehetőségeit kutatta. A kutatás az 1970-es években is tovább folytatódott, ekkor jelent meg Duda és Hart [6] mintázat felismeréssel foglalkozó munkája, valamint a következő évtized elején odáig is eljutottka a kutatók, hogy képesek voltak felismerni egy 40 karakterből álló adatkészletet [7].

A gépi tanulás, mint tudományos kutatási terület, a mesterséges intelligencia utáni kutatás során alakult ki. A mesterséges intelligencia kutatásának kezdetén néhány kutató arra volt kíváncsi, hogy a gépek hogyan képesek adatok alapján tanulni. A kísérletek során számos irányból próbálták megközelíteni a problémát. Olyan szimbolikus módszereket dolgoztak ki, amit később neurális hálóknak kezdtek el hívni, ezeket leginkább perceptronok [] és más modellek alkották.

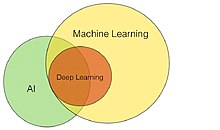
Mindazonáltal a mesterséges intelligencia kutatása egyre inkább a tudás alapú, logikus megközelítés felé tolódott, ami egy törést, távolodást jelentett a gépi tanulás módszereitől. A valószínűségi modellek nagy hibája volt mind elméleti, mind gyakorlati szinten az adatok reprezentálásának és összegyűjtésének kérdése [9]. Többek között ez is volt az oka annak, hogy mind a mesterséges intelligencia kutatások, mind a computer science felhagyott a neurális hálók alkalmazásával.

Ezen történések együttesen vezettek oda, hogy az 1990-es években a gépi tanulás, mint önálló tudományág elkezdjen növekedni és fejlődni. Mindinkább az lett a terület célja, hogy a mesterséges intelligencia elérése helyett olyan problémákkal, kérdésekkel foglalkozzon, hogy hogyan lehet adott problémákat természetes, praktikus módokon megoldani. Ezek hatására kezdett el a gépi tanulás olyan módszereket alkalmazni, mint a statisztika és a valószínűségszámítás [10].



ábra 1: a gépi tanulás a mesterséges intelligencia része, saját kép

Napjainkban találkozhatunk azzal az állítással is, hogy a gépi tanulás továbbra is a mesterséges intelligencia egyik részhalmaza (ábra 2) [12], ugyanakkor olyannal is, hogy a gépi tanulásnak vannak olyan területei is, amelyek nem tartoznak bele a mesterséges intelligencia tudományterületébe (ábra 1) [13].



ábra 2: a gépi tanulás és a mesterséges intelligencia csak részben fedik egymást, forrás: [11]

A mesteréges intelligencia és a gépi tanulás közötti különbséget legjobban talán Judea Pearl fogalmazt meg The Book of Why [14] című könyvében. Ebben azt mondja, hogy míg a gépi tanulás passzív megfigyelések alapján tanul és jósol, addig a mesterséges intelligencia valamilyen ügynököt használ arra, hogy interakcióba léphessen a környezetével, tanulhasson, akciókat hajthasson végre, hogy a célját minél nagyobb arányban érje el.

### Deep learning

Ezen a ponton térnék rá a mélytanulásra, mint témakörre. Az előző pont ábráin láthattuk, hogy a mélytanuló algoritmusok mind mesterséges intelligencia alapú, mind gépi tanuláson alapuló algoritmusokat, módszereket alkalmaznak (ábra 1) (ábra 2). Mint látható a mélytanulás mindkét tudományterületből merít ötleteket, metódusokat, amelyet olyan sikeresen tesz, hogy a mélytanuló algoritmusok napjainkban a vezető módszert jelentik a gépi tanulás területén [2].

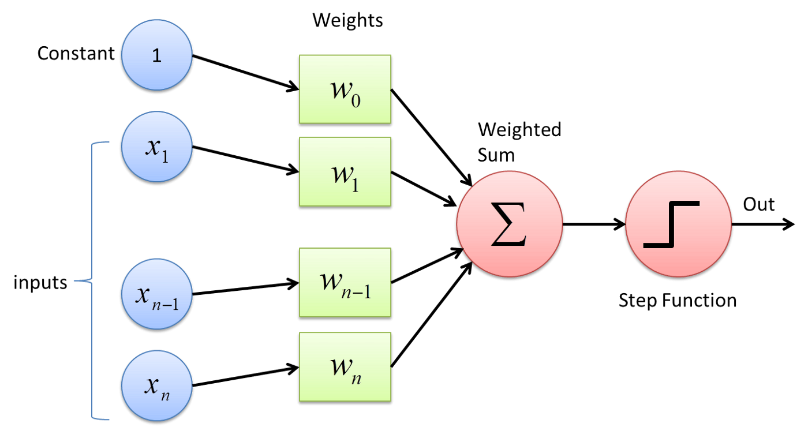
Annak ellenére, hogy a deep learning népszerűsége csak az utóbbi néhány évben kezdett meredek felívelésbe, maga az eljárás és a téma már régebb óta ismert. Az első működő, felügyelt, több rétegű perceptront 1967-ben publikálta Alexey Ivakhnenko és Valentin Lapa [15][17]. A mélytanulás fogalmát 1986-ban Rina Dechter [16][17] vezette be a mesterséges intelligenciával foglalkozó kutatások körébe. A technológia elterjedésének ekkor még elsősorban a rendkívüli adatmennyiség tárolásához és feldolgozásához szükséges tárolási és számítási kapacitás hiánya szabott gátat. Az utóbbi néhány év, évtized robbanásszerű fejlődése az informatikában, a számítási és tárolási kapacitások megnövekedése és az egyre több tárolt adat utat nyitott a deep learning térhódításának.

A deep learning alapjául szolgáló mesterséges neurális hálók ötletét a különböző biológiai rendszerekben megtalálható elosztott kommunikációra és adatfeldolgozásra képes sejtek, pontok inspirálták; gondoljunk csak az emberi agy neuronjaira. Ugyanakkor a szigorúan vett biológia agytól néhány tulajdonságban különbözik is. Míg egy biológiai agy egy élő, változó organizmus, addig egy mesterséges neurális háló statikus és szimbolikus fogalom, amely a működésbeli hasonlóságokra fókuszál.

A mélytanulás kifejezésében szereplő mély melléknév arra utal, hogy egy ilyen struktúrában több, hasonló réteget használunk fel, ezek a rétegek az úgynevezett perceptronok. Egy perceptron képes eldönteni, hogy a számára megadott számokból álló vektor egy adott osztályhoz, csoporthoz tartozik-e vagy sem, amelyet a kimenetén egy bináris számmal jelez. A korai kutatások megmutatták, hogy egy lineáris perceptron nem képes univerzális osztályozásra. Továbbá azt is, ha az ilyen lineáris rétegekből álló hálót ellátjuk egy nemlineáris, aktivációs függvénnyel, akkor az így kapott perceptron már bármilyen kompakt halmazon folytonos függvényt képes tetszőleges pontossággal megközelíteni, feltéve, ha nem korlátozzuk a rejtett réteg szélességét. Ezt nevezik univerzális approximációnak.

EHHEZ KELL VALAMI LINK

TODO



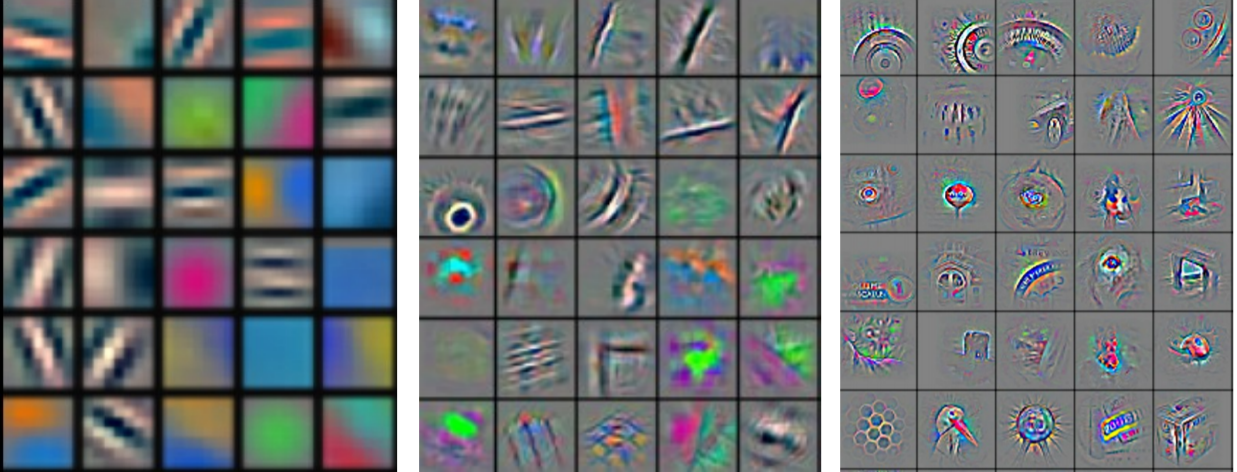
ábra 3: Egy perceptron felépítése, a step function a nemlinearitást hivatott reprezentálni, FORRÁS: []

Ahhoz, hogy ne csak bináris értékeket tudjunk osztályozni nem elég egy darab perceptron. Ahhoz, hogy egy *n* különböző csoportból, osztályból álló adathalmazt fel tudjunk ismerni *n* darab bináris perceptron egymás mellé helyezésére van szükségünk, amelyek közül mindegyik egy-egy osztály, csoport felismeréséért felel.



ábra 4: Egy többrétegű perceptron felépítése, forrás: []

Az ilyen perceptron rétegekből felépülő neurális háló működését szokás úgy bemutatni, mint hierarchikusan egymásra épülő, egyre komplexebb tulajdonságok kinyerésére képes rétegeket. Például képfeldolgozás esetén az alsóbb rétegek felelnek vonalak, görbék felismeréséért, az azt követő réteg ezekből a formákból alkot alakzatokat (kör, négyzet háromszög). Az ezt követő réteg pedig ezekből a formákból alkotja meg a felismerni kívánt alakzatot, és végül dönti el, hogy az megtalálható-e a bemenetként megadott képen (ábra 5).



ábra 5: A különböző absztrakciós szinteket reprezentáló képek deep learning esetén (balról jobbra egyre növekvő absztrakciós szintek) [18]

Természetesen a mai, modern neurális hálók bőven meghaladták az egyszerű perceptronokból felépülő neurális hálókat, azonban az alapelgondolás gyakorlatilag még mindig ugyanaz. A feladattól függően számos különböző struktúra létezik. A felügyelt tanuláson témakörében képfelismerő algoritmusokhoz előszeretettel használnak konvolúciós neurális hálókat (Convolutional Neural Network - CNN) [19], amelyek az információt konvolúciós neurális rétegek segítségével nyerik ki, a konvolúció műveletének segítségével. Ezekből rendkívül változatos struktúrákat lehet felépíteni, amelyek közül néhánnyal én is megismerkedtem és be is fogok mutatni.

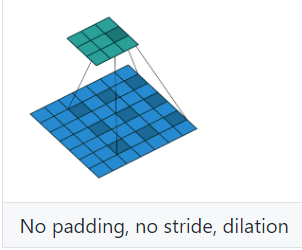
Az előző fejezetben röviden bemutattam a perceptron modellt és azt, hogy ebből az alap építőkőből hogyan és milyen struktúrákat lehet felépíteni. Mindazonáltal egy ilyen többrétegű perceptron modellel nehézkes lenne olyan képosztályozást vagy objektumdetektálást végrehajtani, amilyen feladatokkal a mai neurális hálók szembekerülnek. Éppen ezért egy másik struktúra terjedt el az ilyen jellegű feladatok megoldására, ez pedig a konvolúciós neurális háló, ami konvolúciós rétegekből épül fel. A perceptronokból felépülő modellek hátránya volt az, hogy a térbeliséget nem használta fel, illetve, hogy ezek úgynevezett teljesen összekapcsolt (fully connected) rétegek, azaz két szomszédos perceptron réteg be- illetve kimenete össze van kapcsolva.

Ez a két hátrány a konvolúciós neurális hálóknál nem fordul elő. Egy ilyen konvolúciós réteg a konvolúciós szűrőkhöz hasonlóan működik. Egy ilyen réteg a bemeneti képen, ami általában szürkeárnyalatos vagy színes, N darab különböző konvolúciós szűrőt futtat végig, amelynek eredményeképpen egy N csatornás képet kapunk. Az ezt követő konvolúciós réteg pedig már ezzel az N csatornával dolgozik tovább. Az, hogy mekkora legyen egy ilyen konvolúciós szűrő és hogy hány csatornája legyen az az adott feladattól függő, úgynevezett hiperparaméter. A hiperparaméterek optimalizálásának segítségével tudjuk a tanulást gyorsítani és a háló pontosságát növelni.



ábra 6: konvolúció különböző paraméterek mellett, forrás: [20]

Mint ahogy az a fenti ábrán is látszik, a konvolúciónak számos paramétere van. Ezek közül röviden bemutatnám a legfontosabbakat. A keret (padding) segítségével megadható, hogy legyen-e a kép körül egy üres pixelekből álló keret, ami a konvolúcióba beleszámít-e vagy sem. A lépésközzel (stride) azt lehet megadni, hogy a két konvolúció között hány pixel különbség legyen. Egyes lépésközzel értelmszerűen egy pixelt lépünk odébb, míg kettővel kettő pixelnyit és így tovább. Az utolsó fontos paraméter a dilatáció (ábra 7), ami azt adja meg, hogy a konvolúció széthúzza-e a pixeleket úgy, hogy nem szomszédos pixeleket vesz be az adott konvolúciós ablakba.



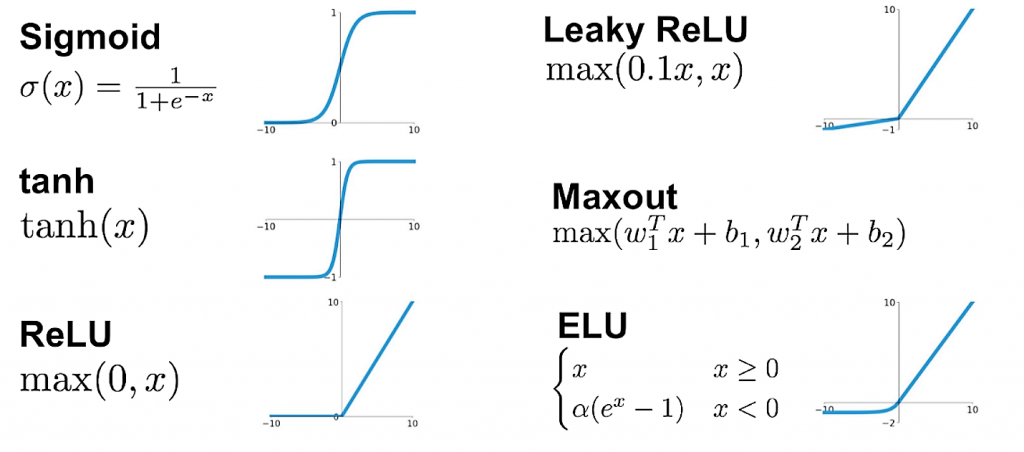
ábra 7: dilatáció konvolúciós szűrő esetén, forrás: [20]

A konvolúció, mint művelet tehát számos hasznos tulajdonsággal bír, különösen olyan térbeli információk kinyerésére, mint amilyenekkel a képfeldolgozás témakörében találkozhatunk. Mindazonáltal nem érdemes csak konvolúciós rétegeket alkalmazni, ha egy modellt szeretnénk felépíteni, ugyanis minél több ilyen konvolúciós réteget helyezünk egymás mögé, annál nagyobb lesz a konvolúció kimenetén kapott struktúra mérete. Ennek elkerülése végett érdemes néhány rétegenként csökkenteni a tömb méretét. Ezt megtehetjük úgy, hogy a konvolúció lépésközét növeljük, vagy beiktatunk egy tömörítő, úgynevezett pooling réteget is. Egy ilyen művelet során valamilyen módszer szerint a tömb aktuális részét egy adott számmal helyettesíti. Ez a módszer legtöbbször vagy az ablakban szereplő értékek átlagának számítása, vagy a maximális érték választása (21).



ábra 8: max (balra) és átlagoló (jobbra) pooling, forrás: [21]

Az utolsó fontos komponens a konvolúciós neurális háló struktúrákban az aktivációs réteg. Ilyen aktivációs rétegek általában egy konvolúciós és lineáris réteget követően helyezkednek el, ugyanis mind a konvolúció mind a lineáris réteg lineáris műveleteket alkalmaznak, ezért fontos beékelni nemlinearitást a perceptron bemutatásánál említett okokból. Amikor elkezdtek konvolúciós neurális rétegeket alkalmazni előszeretettel használtak szigmoid vagy hiperbolikus tangenst (ábra 9). Azonban ezek deriváltja az értelmezési tartományuk szélén alacsony, ami ahhoz vezet, hogy a deriváltak nullához fognak konvergálni emiatt pedig elrontják a tanítást. Emiatt mindinképp a ReLU (Rectified Linear Unit) kezdett elterjedni (ábra 9), amely kiküszöbölte ezt a problémát és a deriváltjának számítása is gyorsabb. Azóta a ReLU aktivációt tovább finomították, hogy még hatékonyabban működhessen (ábra 9).



ábra 9: különböző aktivációs függvények – balról lefelé, oszloponként: szigmoid, hiperbolikus tanges, ReLU, Leaky ReLU, Maxout és ELU, forrás: [22]

Az eddig bemutatott rétegek felhasználásával már fel lehet építeni egy olyan konvolúciós neurális háló struktúrát, amivel már nagy pontossággal lehet megoldani könnyebb feladatokat. A következő alfejezetben be fogom mutatni, hogy jelenleg milyen objektumdetektáló neurális hálókat használnak, hogy néz ki ezek szerkezete, milyen előnyeik és hátrányaik vannak.

## Give introduction into narrow field (e.g: object detection, reinforcement learning) 5 pages

A házi feladatom során én is felügyelt tanulást hajtok végre, ezért a következőkben ezt fogom bemutatni. A felügyelt tanulás a deep learning egyik alfaja, amelynek során a DNN-t (Deep Neural Network – mély neurális háló) egy nagy adatbázis segítségével tanítjuk meg adott feladatra. A képi adatbázis minden egyes képéhez tartozik egy címke is, amin azt jelezzük a neurális hálónak, hogy mi az adott bementre elvárt, a háló által adandó kimenet. A neurális háló arra lesz képes, hogy meghatározza, hogy a tanítás során megadott osztályok közül jelen van-e a képen valamelyik.

A feladat az, hogy az elvárt kimenetek függvényében a neurális háló minél nagyobb pontossággal tudja eltalálni azt, hogy az aktuálisan a bemenetére adott kép melyik osztályba tartozik. A háló kezdetben valamilyen véletlen kimenetet ad, hiszen nincs megtanítva a helyes kimenetre. Ezt valamilyen tanító függvény segítségével lehet megtenni. A tanítás során az elvárt kimenet és a háló által adott kimenet között valamilyen különbségképző algoritmussal adjuk meg (például Cross Entropy Loss, MSE Loss) (7), hogy mekkora hibát vét a háló. A hiba alapján egy optimalizáló algoritmus segítségével (például Adam optimization) (8) számítjuk ki, hogy hogyan változtassuk a háló súlyait, hogy minél pontosabb eredményt adjon. Ezt a backpropagation segítségével lehet elérni, amely során a háló neuronjait összekötő kapcsolatok súlyait (és ofszetjeit (bias)) az alapján módosítjuk, hogy mekkora mértékben tér el az elvárt kimenettől. Ezt a gradiens módszerrel tehetjük meg, amely során a parciális deriváltakat számítjuk ki a láncszabály segítségével és annak tükrében módosítjuk a súlyokat és ofszeteket.

A modern felügyelt tanulás alapú neurális hálók (DNN) is ezt a struktúrát követik azonban számos új metódus született annak érdekében, hogy minél jobb eredményt érjenek el. A képfeldolgozás témakörén belül, mint már említettem, a konvolúciós neurális hálók váltak rendkívül elterjedté. A CNN-ek a konvolúció műveletével nyerik ki az információt a feldolgozandó képekből. Az első konvolúciós neurális hálót Kunihiko Fukushima alkotta meg 1980-ban (5). A kép egy adott részletén a konvolúciót elvégezve, abból magasabb szintű információ nyerhető ki, ezzel pedig hierarchikus szinteket lehet felépíteni, hogy valamilyen absztrakt eredményt érhessünk el.



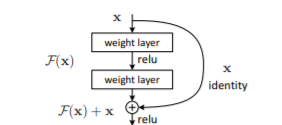
Konvolúciós neurális háló struktúra (24)

Előnye egy lineáris neurális hálóval szemben, hogy az egyes vektorelemek (pixelértékek) egymáshoz képesti pozícióját képes felhasználni, míg ezt egy lineáris réteg nem volt képes kezelni, hiszen az egy egyszerű sorvektorként dolgozta azt fel. További előnye még egyéb algoritmusokhoz képest, hogy az egyes képeket relatíve kicsi előfeldolgozással tudja fogadni. Hátránya, hogy az egyes részletek egymáshoz képesti helyzetét még így sem tudja teljesen jól kezelni (például egy arc esetén a szem, száj, orr egymáshoz képesti helyzetét nem tudja jól felismerni). További hátránya, hogy numerikusan instabil lehet és csak olyan dolgokat képes felismerni, amelyre a tanítás során megtanítottuk (ez minden felügyelt tanulás alapú neurális hálóra igaz). Nem képes tehát saját döntéseket, saját címkéket generálni, olyan információt alkotni, amelyet a megalkotói ne tudtak volna.

A félév során végzett munkám keretében számos modern neurális háló struktúrával is megismerkedtem, néhányat ezek közül ki is próbáltam, mielőtt a végleges algoritmust elkezdtem volna feldolgozni és megérteni. A megismert struktúrák között szerepel a ResNet (9) és az Inception (10)(11), melyek közül a ResNettel foglalkoztam részletesebben.

### ResNet:

Az általános egymás után helyezett konvolúciós rétegekkel ellentétben a ResNet rendelkezik úgynevezett reziduális kapcsolattal, amely egy előre csatoló ágat jelent a struktúrában. Ezzel elérhető, hogy a konvolúció műveletek miatt elvesző információkat visszanyerhessük, így a háló teljesítménye növelhető. Az eddigi neurális hálókkal ellentétben a ResNet képes hasonlóan jó eredményt produkálni sokkal kevésbé mély struktúrával, ezáltal ugyanahhoz a művelethez kevesebb számítási kapacitást igényel. Ugyanakkor képes arra is, hogy egy olyan mély struktúra esetén is megőrizze a numerikus stabilitását, ahol már egy klasszikus, reziduális rétegekkel nem rendelkező háló nem lenne képes, mindezt úgy, hogy még így is kevésbé komplex.



Reziduális építőegység (9)

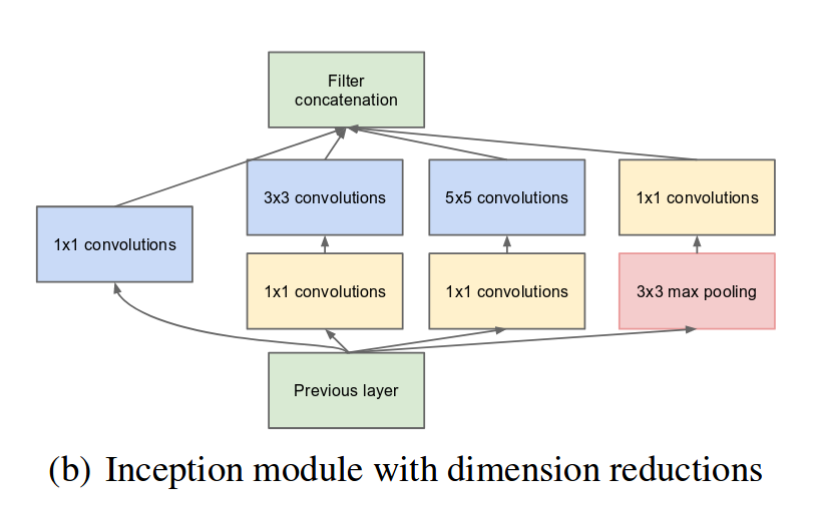
### Inception net:

A másik modern konvolúció neurális háló struktúra, amivel tüzetesebben foglalkoztam az Inception Net (GoogleNet). Egy képen szereplő osztály detektálásakor a számunkra fontos képrészlet rendkívül változatos lehet a méretét tekintve. Éppen ezért a megfelelő kernelméret kiválasztása rendkívül nehéz lehet. Továbbá a rendkívül mély neurális hálók hajlamosabbak az overfitting jelenségére (a tanító adatsoron remek pontosság, de a validációs és tesztadatokon rossz eredmény), valamint a számítási kapacitást tekintve is drágábbnak minősülnek.



A kép különböző részét kitöltő, ámde ugyanazt az osztályt (kutya) ábrázoló képek

Az Inception Net struktúra erre kínál megoldást. Az elgondolás alapja az, hogy egy adott szinten nem egy konvolúciót hajt végre, hanem egyazon szinten több különböző kernel mérettel is elvégzi a konvolúciót (továbbá egy max poolingot is végrehajt), majd az elvégzett konvolúciók kimenetét konkatenálja és ezt adja tovább a következő ilyen rétegnek. Ahhoz, hogy a számítási kapacitást csökkentse, a réteg egy extra 1x1-es konvolúciót is végrehajt a rétegen belül, ezzel limitálva a bemeneti csatornaszámot. Noha ez ellentmondásosnak tűnhet, hiszen pluszműveletet jelent, egy 1x1-es konvolúciót sokkal kevésbé költséges elvégezni, mint egy 5x5-s konvolúciót.

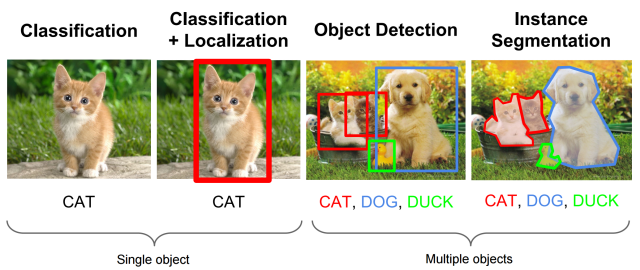


Az inception háló (Inception v1 – GoogleNet) építőegysége dimenzióredukcióval ellátva (11)

A teljes neurális háló ilyen építőegységekből épül fel, természetesen további kiegészítésekkel, amelyekre most nem térek ki, mert nem képezik szerves részét a beszámolómnak. Az struktúrát a későbbiekben tovább is bővítették, finomították, így született meg az Inception v2 és v3.

## Objektum detektálás

Az előző bekezdések során ismertetett neurális hálók objektum osztályozó neurális hálók voltak, ami azt jelenti, hogy rendkívül jó pontossággal képesek megmondani, hogy egy adott osztály szerepel-e a képen vagy sem. Azonban vannak jelentős hátrányaik, amelyek az én témám esetében nagy jelentőséggel bírnak. Nem képesek kezelni azt, ha egy adott képen több osztályból is szerepelnek objektumok, valamint arra sem képesek, hogy a képen belül meghatározzák, hogy hol helyezkednek el ezek az objektumok.



Különböző objektum meghatározási módszerek

A témámhoz tehát egy olyan neurális hálóra struktúrára volt szükségem, amely képes egyszerre több objektumot felismerni és azok helyzetét meghatározni. A félév elején kiválasztott algoritmus a YOLO objektum detektálásra képes struktúra volt, ugyanakkor a félév során más, hasonlóan működő struktúrákkal is megismerkedtem. A következőkben bemutatom az egyes objektum detektáló módszereket, néhány példát, ezután pedig a YOLO architektúrát és működését fogom bemutatni részletesebben.

A különféle objektum detektáló algoritmusokat két csoportba lehet sorolni. Az első csoport a klasszifikáción alapuló algoritmusok. Ezeket az algoritmusokat működésük szempontjából két lépésre lehet bontani. Első lépésként kiválasztanak a képen egy, az algoritmus alapján érdekesnek talált régiót, második lépésként pedig ezeket a régiókat osztályozzák valamilyen CNN segítségével. Ez a módszer nem kifejezetten gyors, hiszen minden kiválasztott régión le kell futtatni az osztályozó algoritmust, ugyanakkor pontosabbak lehetnek, mint a másik alcsoport. Ehhez a kategóriához tartozik például a régió alapú neurális háló (Region-based convolutional neural network (RCNN)) (15) és ennek a továbbfejlesztett verziói, a Fast-RCNN (16), Faster-RCNN (17), valamint a legújabb fejlesztés, a Mask-RCNN (20). Egy másik példa erre a típusra a RetinaNet (18).

A másik csoport a regresszión alapuló algoritmusok. Ezek az algoritmusok ahelyett, hogy kiválasztanák a kép egy adott részét és azon végeznék el az osztályozást, az egész képen próbálják meg előrejelezni az objektumosztályokat és az azt övező bounding box-okat, mindezt egy egyszeri futtatás során. A két leginkább elterjedt algoritmus ebben a kategóriában a YOLO (You Only Look Once) (12)(13) algoritmus, illetve az SSD (Single Shot Multibox Detector) (19). Ezeket az algoritmusokat elsősorban valós idejű alkalmazások során használják, hiszen a pontosságbeli hátrányukat a sokkal gyorsabb futási sebességükkel kompenzálják.

A fenti megfontolások alapján tehát egy regresszión alapuló, valós időben futtatható algoritmusra volt szükségünk, a választásunk pedig a YOLO algoritmusra esett.

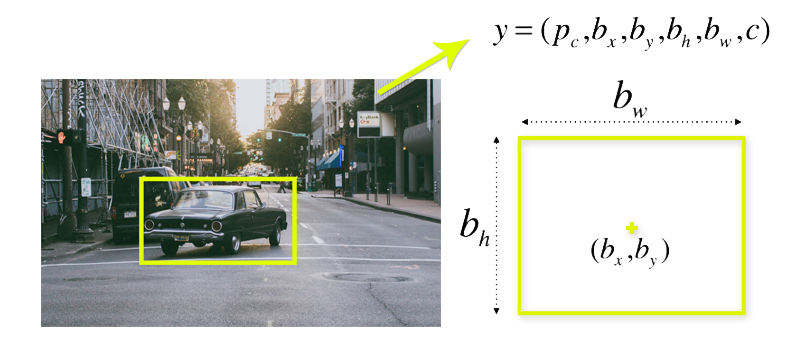
## Describe chosen algorithm and its alternatives 5 pages

A YOLO algoritmus (You Only Look Once) egy hatékony, valós időben futó objektum detektáló algoritmus, amelynek első verzióját Joseph Redmon és társai publikáltak 2015-ben. (12) Az algoritmus egy objektum detektálásra képes metódus, tehát egy képen több osztály több objektumát is képes észlelni, és azok köré egy úgynevezett bounding box-ot (körülvevő keretet/ dobozt) helyezni. Később több, további verzióját, továbbfejlesztését is publikálták az algoritmusnak, a legutóbbi a YOLOv3 (13), amit 2018-ban publikáltak, a következőkben ennek a működését fogom bemutatni.

Mint a legtöbb modern DNN a YOLO is konvolúciós neurális háló (CNN), ezen belül egy teljesen konvolúciós neurális háló (FCN), ami azt jelenti, hogy csak konvolúciós rétegekből épül fel. Összesen 53 konvolúciós réteget tartalmaz, skálázott (strided) rétegekkel és előrecsatoló (reziduális) kapcsolatokkal.

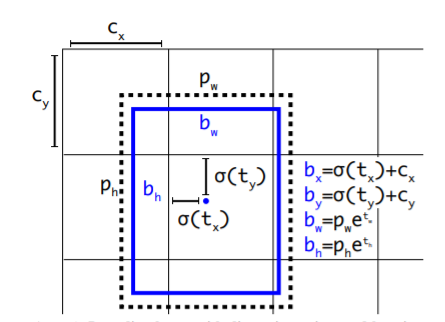
Az eddig megismert struktúrák esetén, a konvolúciós rétegek által megtanult kimeneti valószínűség vektort valamilyen lineáris réteg adja ki. A YOLO esetén ez nem így van, a detektálást egy 1x1-es konvolúciós réteg végzi el, tehát ugyanakkora lesz a kimeneti „kép”, mint az azt megelőző réteg. Ennek a kimeneti képnek a csatornaszáma összesen (B \* (5 + C)), ahol B az adott cella által meghatározott bounding box-ok száma (YOLOv3 esetén ez a szám 3), C pedig az adatbázisban szereplő osztályok száma. Több bounding box-ot azért használunk egy cellán belül, hogy több objektumot is észlelni tudjunk, amelyek ugyanabba a cellába esnek.

A végső célunk az, hogy egy adott osztályba tartozó objektumot észleljünk, továbbá azt is meghatározzuk, hogy az hol helyezkedik el a képen. Ezt a már említett bounding box-ok segítségével tehetjük meg, amit négy paraméter segítségével lehet leírni. Ezek a bounding box szélessége (bw), magassága (bh), valamint a bounding box középpontjának koordinátái (bx, by). Az algoritmus ezen felül kimenetként megadja, hogy mekkora valószínűséggel van jelen egy objektum az adott bounding box-ban, illetve, hogy az mekkora valószínűséggel egy adott osztály tagja.



A YOLO algoritmus bounding boxának paraméterei, valamint a kimeneti vektor elemei (13)

Az algoritmus a képet azonos méretű cellákra osztja (a képtől és a leskálázástól függően), és minden egyes cellára meghatározza a fent említett kimenetet. Azt várjuk el, hogy minden egyes cella detektáljon egy ilyen objektumot abban az esetben, ha a bounding box-ának középpontja az adott cellába esik. Annak elkerülésének érdekében, hogy az adott koordináta ne lóghasson ki a cellából, a bounding box-ot meghatározó paramétereken egy transzformációt hajtanak végre.

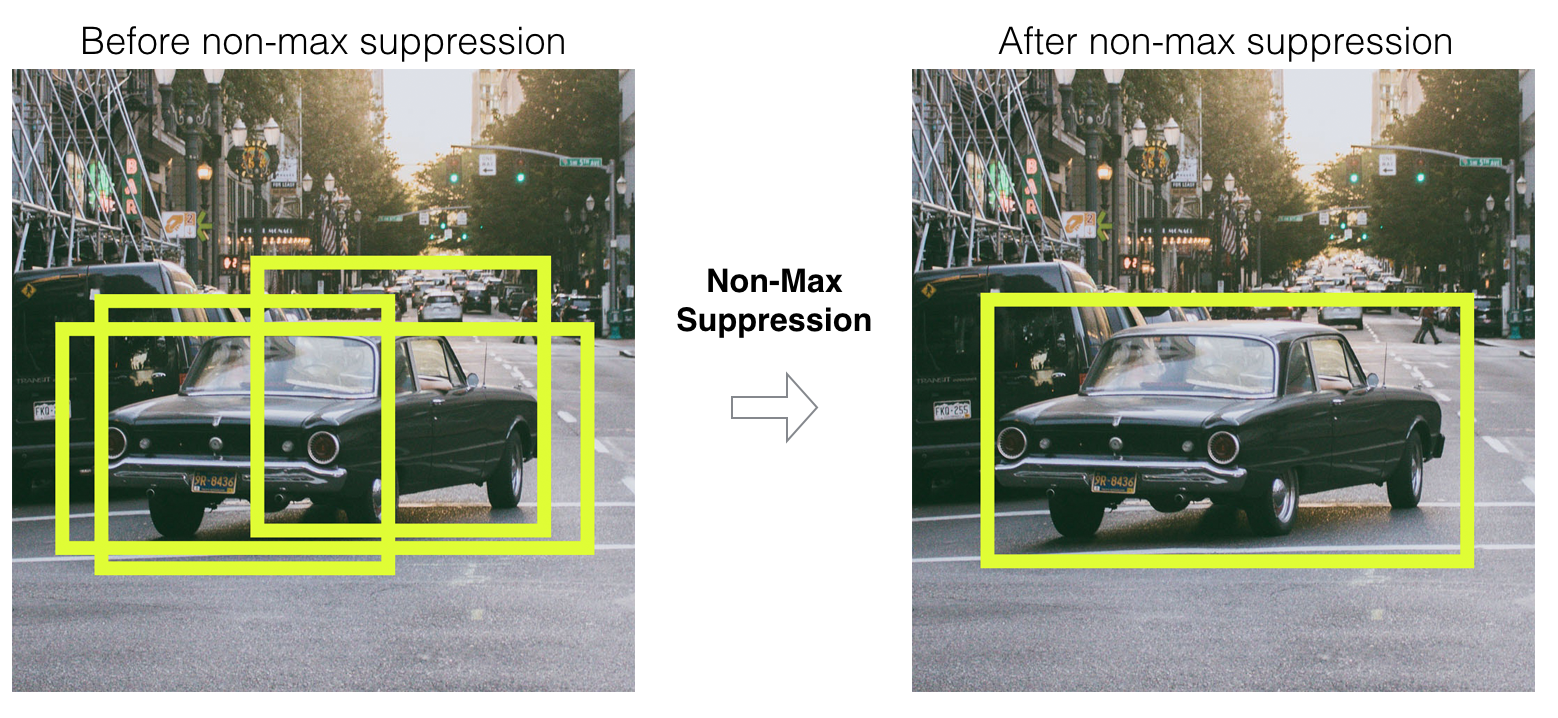


A YOLO algoritmus bounding box leképezés transzformációja (12)

A fenti kép alapján tehát a háló a tx, ty, tw és th paramétereket határozza meg és a fent látható leképezés segítségével kapjuk meg a végleges, fent részletezett paramétereket.

A YOLOv3 algoritmus 3 különböző méretű képen végzi el a detekciókat. Az algoritmus futása során a különböző szintű konvolúciók elvgzése során teszi ezt meg. Nézzük a következő példát, ahol a bemeneti kép 416x416 pixeles, és a három leskálázó réteg méretei: 32,16 és 8. Ez azt jelenti, hogy a kimeneten megkapott képek cellaszámai: 13x13, 26x26 és 52x52. Az algoritmus minden egyes szinten 3 bounding boxot határoz meg minden egyes cellához, ami azt jelenti, hogy összesen ((52 x 52) + (26 x 26) + (13 x 13)) x 3 = 10647 bounding box-ot határoz meg.

Ez a szám rendkívül nagy, természetes nem lesz ennyi objektum a képen. Ezt a számot az algoritmus két lépésben csökkenti le. Elsőként egy bizonyos küszöbszinti alatti valószínűséggel rendelkező detekciókat törli, majd ezt követően Non-Maximum Suppression-t (NMS) hajt végre és így kapjuk meg a végleges, az adott objektumhoz tartozó detekciót.



NMS a YOLO algoritmus során (13)

### Make sure you do some pros/cons

# Specification and design (8-10 pages):

## Describe the actual task in detail

## Make sure to mention constraints, limitations, requirements

## Which of the algorithms described in the Related Work chapter we chose and why

## What frameworks are available and how are they different and why we chose the one we chose

## Software design (if there is one)

# Development (chapters 5-6 should be 15-25 pages):

## What Programming language/IDE/Environment we used

## Dataset creation (if there is one)

## Describe implementation of modules

## Describe algorithms/neural net structure

## Describe different attempts

# Experimental Results:

## Describe experiment methodology

## Describe test cases

## Describe metrics used for evaluation

## Describe dataset (if there is one and it wasn't created)

## Describe test cases

## Evaluate results

# Conclusion (1-2 pages):

## Give general conclusion (what we did and why is it good)

## Evaluate results quickly

## Describe opportunities for further research/improvement

# Források, hivatkozások

1. Tom Mitchell: Machine Learning (1997): <http://www.cs.cmu.edu/~tom/mlbook.html>
2. Ethem Alpaydin (2020). Introduction to Machine Learning (Fourth ed.).
3. Bishop, C. M. (2006), Pattern Recognition and Machine Learning, Springer, ISBN 978-0-387-31073-2
4. Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. 3 (3): 210–229 <https://ieeexplore.ieee.org/document/5392560>
5. Nilsson N. Learning Machines, McGraw Hill, 1965.
6. Duda, R., Hart P. Pattern Recognition and Scene Analysis, Wiley Interscience, 1973
7. S. Bozinovski "Teaching space: A representation concept for adaptive pattern classification" COINS Technical Report No. 81-28, Computer and Information Science Department, University of Massachusetts at Amherst, MA, 1981. <https://web.cs.umass.edu/publication/docs/1981/UM-CS-1981-028.pdf>
8. Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 978-0-07-042807-2.
9. Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. ISBN 978-0137903955.
10. Langley, Pat (2011). "The changing science of machine learning". Machine Learning. 82 (3): 275–279. doi:10.1007/s10994-011-5242-y.

1. ["rasbt/stat453-deep-learning-ss20"](https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L01-intro/L01-intro_slides.pdf) (PDF). *GitHub*.
2. *Garbade, Dr Michael J. (14 September 2018).*[*"Clearing the Confusion: AI vs Machine Learning vs Deep Learning Differences"*](https://towardsdatascience.com/clearing-the-confusion-ai-vs-machine-learning-vs-deep-learning-differences-fce69b21d5eb)*. Medium. Retrieved 28 October 2020.*

1. ["Chapter 1: Introduction to Machine Learning and Deep Learning"](https://sebastianraschka.com/blog/2020/intro-to-dl-ch01.html). *Dr. Sebastian Raschka*. 5 August 2020. Retrieved 28 October 2020.
2. *Pearl, Judea; Mackenzie, Dana.*[*The Book of Why: The New Science of Cause and Effect*](https://www.basicbooks.com/titles/judea-pearl/the-book-of-why/9780465097609/)*(2018 ed.). Basic Books.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9780465097609*](https://en.wikipedia.org/wiki/Special:BookSources/9780465097609)*. Retrieved 28 October 2020.*
3. (1): A. G. Ivakhnenko, Valentin Grigorʹevich Lapa: Cybernetics and forecasting techniques (1967) <https://books.google.hu/books?id=rGFgAAAAMAAJ&redir_esc=y>
4. (2): Rina Dechter: Learning While Searching in Constraint-Satisfaction-Problems (1986): <https://www.researchgate.net/publication/221605378_Learning_While_Searching_in_Constraint-Satisfaction-Problems>
5. (3): Juergen Schmidhuber: Deep Learning (2015): <http://www.scholarpedia.org/article/Deep_Learning>
6. (4): Matthew D Zeiler, Rob Fergus: Visualizing and Understanding Convolutional Networks (2013): <https://arxiv.org/abs/1311.2901>
7. (5): Kunihiko Fukushima: Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position (1980): <https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf>
8. <https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md>
9. <https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451>
10. <https://7-hiddenlayers.com/deep-learning-2/>
11. (6): Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio: Generative Adversarial Nets (2014): <https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
12. (7): Cross Entropy Loss, MSE Loss: <https://pytorch.org/docs/stable/nn.html#loss-functions>
13. (8): Diederik P. Kingma, Jimmy Ba: Adam: A Method for Stochastic Optimization (2017): <https://arxiv.org/abs/1412.6980>
14. (9): Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Deep Residual Learning for Image Recognition (2015): <https://arxiv.org/pdf/1512.03385.pdf>
15. (10): Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich: Going deeper with convolutions (2014): <https://arxiv.org/pdf/1409.4842v1.pdf>
16. (11): Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna: Rethinking the Inception Architecture for Computer Vision (2015): <https://arxiv.org/pdf/1512.00567v3.pdf>
17. (12): Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: You Only Look Once: Unified, Real-Time Object Detection (2015): <https://arxiv.org/pdf/1506.02640.pdf>
18. (13): Joseph Redmon, Ali Farhadi: YOLOv3: An Incremental Improvement (2018): <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
19. (14): <https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>
20. (15): Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik: Rich feature hierarchies for accurate object detection and semantic segmentation (2014): <https://arxiv.org/pdf/1311.2524.pdf>
21. (16): Ross Girshick: Fast R-CNN (2015): <https://arxiv.org/pdf/1504.08083.pdf>
22. (17): Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (2016): <https://arxiv.org/pdf/1506.01497.pdf>
23. (18): Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár: Focal Loss for Dense Object Detection (2018): <https://arxiv.org/abs/1708.02002>
24. (19): Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg: SSD: Single Shot MultiBox Detector (2015): <https://arxiv.org/abs/1512.02325>
25. (20): Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick: Mask R-CNN (2017): <https://arxiv.org/abs/1703.06870>

Chapters 4-5 can be merged into a "Specification and Design" in case of an algorithm-oriented thesis.

Chapters 5-6 can be merged into a "Design and Development" in case of a hardware-oriented thesis.

**Do:**

* **Always** number your tables, figures and equations.
* **Always**refer to your figures from the text. This tells the reader when to look at them (they might be on the next or the previous page).
* **Always** explain colors and other markings in figures. Preferably in the caption.
* When using abbreviations, expand them for the first time.
* At the start of every chapter, write a paragraph listing what you will talk about in the chapter. This helps the reader to see the underlying structure.

**Don't:**

* **Don't ever**, ***ever***cite Wikipedia. Wikipedia is really good, but it is a collection of references. All it takes is to scroll down to the bottom of the page to find the sources it is citing.
* **Don't** cite lecture slides or notes. Find the paper detailing the algorithm that you learned about in the lecture.
* **Don't** insert equations as images. Sure, it is easy to cut them from the slides using Snipping Tool, but there is a special place in hell for people who do this.
* **Don't** put code, or explain code in the report/thesis. Unless the specific code that you wrote is absolutely brilliant and essential to the value of your thesis. If you want your thesis to be reproducible, put your code on GitHub.